Data Centers

Today's lecture is about datacenter. Datacenter becomes a very popular term after many giant network company built their own datacenter. So today we'll demystify datacenter, starts from the origin of datacenters, dig into the architecture in datacenter from network's perspective and also the congestion control in datacenter. In short, we'll use 2 papers - VL2 [1] and DCTCP [2] to explain what's the difference between datacenter networks and general networks.

1. The origin of datacenters

1.1 Large scale web search

The origin of datacenter can be traced back to the rapid increase in the size of the world wide web in the early 2000s. Search engines of the time, including Google, which was incorporated in 1998, had to cope with searching an ever-growing world wide web. In a very short time, single server was not capable of storing all the data and it must be split to many different machines which is the early prototype of datacenter.

1.2 Definition and hardware

There's not a formal definition of datacenter, some definition includes requirements of several redundancy hardware or spare devices. From network's perspective, datacenter is a private network controlled by one economic entity, ex: Google or Facebook's datacenters. There are two basic components in datacenters, one is computing power and the other is network. Usually one datacenter will have about O(100K) servers and O(10K) switches. Right now, there're about 100 datacenters around the world and is still growing. Each one is about 5~10 football field size.

1.3 Difference between datacenter and supercomputer

The role of datacenter and supercomputer are really similar, they both use a lot of servers(clusters) to achieve the task which is hard(or slowly) be done by single server but their consideration are different. For supercomputer, it tries to maximize the performance but datacenter considers on performance / cost. It doesn't mean cost is not important in supercomputer, just its focus is more on computing power since the problem it tried to solve is usually really difficult, ex: weather prediction.

1.4 Cloud computing: computing as a service

Cloud computing is a term borned with datacenter. It's like a service, for example, datacenter's owner like Amazon, Google and Microsoft rent their network, storage, computing power to end users. From owner's perspective, they can make money by leasing out their idle resources. From end user's perspective, they don't have to maintain static hardware's cost(ex: servers, bandwidth at IDC), developers or technology company can focus more on their service and adjust their hardware requirements quickly. Cloud computing just provides flexibility at both side.

1.5 Summary

After briefly elaborating what the datacenter is, we will go deeper two specific concepts in datacenter related to network. One is datacenter's network topology, we use VL2 as reference. The other is datacenter's congestion control protocol, here we use DCTCP as reference.

2. Datacenter's topology

The goal of datacenter is to let any server inside the datacenter to communicate with each other as though they have dedicated high capacity link between them. The datacenter is like a big switch connecting N servers but with far fewer than N^2 links or a switch with N ports - when N is large that's not possible. So how does it actually work? The idea goes back to Clos Network which tried to solve similar problem in telephone. But before jumping into the modern architecture, let's understand the issue in the conventional datacenter.

2.1 Datacenter's challenge before

Check VL2's paper's Figure 1, it's quite easy to see what's the issue in this kind of design. First, this hierarchical design will oversubscribe the network, from 1:5 to the root about 1:240 which deeply affects server-to-server's capacity. Second, the reliability is bad because each server only has one redundancy. There're also many drawbacks of this design, like fragmentation of resources and traffics affect each other, etc.

2.2 VL2's objectives

Microsoft first observed these problems in their datacenter and provides the ideas.(Not so sure if it's really the first one because like Google, Facebook would publish their infrastructure many years after their survey) And there's a core concept of the whole design - using original servers with progressive deployment. It's preferable to leverage original software/hardware than investing the new one - save cost. There are three objectives in VL2's design:

a. Uniform high capacity - Maximum rate from server to server and should be independent of network topology.
b. Performance isolation - Traffic of services should not be affected by each other.
c. Layer-2 semantics - Easily assign any server to any service which is crucial to easily move customer's deployments to cloud and also keep link-layer broadcast feature.

These will make datacenter agile and have better fault tolerance when there's any error in software/hardware stack.

2.3 Measurements and main idea

Microsoft run real measurements on their datacenter and also interview many related engineers to get the insight. They have two major results. First, there's a huge volatility in datacenter's traffic matrices(TM) which makes traffic engineering hard to do. Second, the hierarchical topology is really unreliable since there's only 1:1 redundancy. After studying all these patterns, they come to the core design philosophy - randomness. The main components of VL2 are in the following:

a. Using Valiant Load Balancing(VLB) to cope with traffic volatility by destination independent traffic.
b. Using existing switch's IP routing and forwarding technologies.
c. Using directory system to maintain the mapping between application addresses(AA) and location addresses(LA).
d. Embracing end systems by using VL2 agent at each server to perform find-grained ACL to control communication.

2.3 Topology

Let's find out what's the real implementation in VL2. In [3] Figure 1, assume a k-port switch that can connect to k servers. First, create a leaf layer of k k-port switches. For each switch in the leaf layer, we connect k / 2 ports (i.e., half the ports) to k / 2 servers. Second, create a spine layer of k / 2 k-port switches. We connect each of the remaining k / 2 ports on each leaf-layer switch to all k / 2 switches in the spine layer. Typically k is around 100.

Clearly, this design provides better redundancy, if one server is down, it won't influence the traffic by 50%, it will only affect 1/k portion of the traffic. And also, we can build this architecture by using low end components.

2.4 Details

2.4.1 Addressing

VL2 separates names from locations so they can retain even after VM migration. During packet forwarding, encapsulates application's packet by outer IP layer header with source switch's IP using VL2's agent. Decapsulation is on the receiver side. And about address resolution, VL2 uses a Directory System to map application addresses(AA) to location addresses(LA).

2.4.2 Valiant Load Balancing(VLB)

VLB randomize each flow by randomly choose path by using Equal cost multi-path routing(ECMP). This mechanism is proved hotspot free and there are other load balancing algorithm been invented since then, like CONGA, PRESTO, HULA, LetFlow, etc. which can replace ECMP's role.

2.4.3 Directory System

Basically it's a two tier system. Tier 1 for read optimized cached servers and Tier 2 for write operation. The system should be eventually consistent but would take a small time.

2.5 Summary
After implementing the system, VL2 evaluates the system by all-to-all shuffle(like MapReduce jobs) and get uniform high capacity. Also, it provides fairness by randomly spliting traffic and performance isolation. What's not so clear is the directory system, the paper didn't talk about where exactly it is located and it's also hard to implement this system. After VL2 was published, most datacenters now use VL2 style topology and also employ some load balancing technique like ECMP.

3. Congestion Control in Data Center(DCTCP)
Datacenter host diverse applications, mixing different workloads that require small predictable latency with others requiring large sustained throughput. In this environment, it's hard to satisfy both low delay and high throughput constraints simultaneously. For example, if one tries to improve the delay by reducing the queue size, then packet drops will cause cwnd(congestion window) reduced and then reduces the throughput. On the other hand, increasing queue size can reduce the packet drop but the delay will increase. This dilemma stimulates the invention of DCTCP.

3.1 TCP and DCTCP
In this kind of mixing environments, TCP has three major impairments:
a. Incast. Caused by Partition/Aggregation scenario, there might be a large number of synchronized short flow hit the same queue which let some flow be timeouts and needs to be retransmitted.
b. Queue buildup. Since datacenter is mixed with long and short flows, when long flows occupied the queue, it will influence short flow's latency.
c. Buffer pressure. Due to the shared memory pool, after long flow build up queues, it will reduce the buffer space to absorb bursts from Partition/Aggregation traffic and cause packet loss and timeouts.
DCTCP tries to fix these issues. There are three requirements it wants to meet simultaneous: 1. High burst tolerance, 2. Low latency and 3. High throughput .

3.3 measurements
In DCTCP's paper, they measured from 6000 production servers. The collected logs are from application level, sockets level and selected packet level. Total data size is about 150TB(compressed)/month.

3.4 DCTCP algorithm
The main idea of DCTCP's algorithm can break down to two parts:
a. On switch side, mark the packet based on specific queue length. This mechanism can feedback quickly to better deal with burst. About marking mechanism, it used Explicit Congestion Notification(ECN) which is already available in commodity switch.
b. On sender side, react in proportion to the extent of congestion, not to its presence. This mechanism is like TCP, but reducing window size based on fraction of marked packets.
Let's make a simple example to see the difference behavior between TCP and DCTCP.

ECN Marks	TCP	DCTCP
1 0 1 1 0 0 1 1 1 1	Cut window by 50%	Cut window by 35%
0 0 0 0 0 0 0 0 0 1	Cut window by 50%	Cut window by 5%

How to change cwnd:
Each RTT,
F = # of marked ACKs / # of total ACKs
alpha = (1 - g) * alpha + g * F
cwnd = cwnd * (1 - alpha / 2) g is the weight given to new samples against the past, from 0 ~ 1

3.5 Summary

Let’s revisit three requirements and see why DCTCP works.

1. High burst tolerance: DCTCP can provide larger buffer space so burst can fit in. Also, the aggressive marking can let sender react quickly before packets are dropped.
2. Low latency: DCTCP occupies small buffer which can decrease queueing delay.
3. High throughput: Since it averages ECN to control its window size, it will make rate adjustment be more smooth and let congestion window(cwnd)’s variance be low.

The beauty of DCTCP is it just used the commodity switch to achieve its amazing result and the modification is quite simple(30 line of codes).

Reference:

[1] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. VL2: A Scalable and Flexible Data Center Network. In SIGCOMM, 2009.
[2] Alizadeh, Mohammad, et al. “Data center tcp (dctcp).” ACM SIGCOMM computer communication review. Vol. 40. No. 4. ACM, 2010.
[3] Note of Datacenter in course CSCI-UA.0480-009: Computer Networks, Anirudh Sivaraman.